The Motor Industry Software Reliability Association

MISRA c/o Electrical Group, MIRA, Watling Street, Nuneaton, Warwickshire, CV10 0TU, UK.
Telephone: (024) 7635 5290. Fax: (024) 7635 5070. E-mail: misra@mira.co.uk Internet: http://www.misra.org.uk

Report 8

Human Factors in Software Development

February 1995

PDF version 1.0, January 2001

Acknowledgements

The following were contributors to this report:

Nigel Barrett, Delco
Josh McCallin, Delco
David Newman, Ford Motor Company Ltd
Heather Storey, MIRA

Summary

Ergonomics, or Human Factors, refers to designing for human use and has arisen from the fact that technological developments have focused attention on the need to consider human beings in such developments. As a discipline it focuses upon an individual's interaction with products, equipment, environments etc that are used in work and everyday living. As a result it seeks to change the things people use and the environment within which they use these things to better match the capabilities, limitations and needs of people.

As the discipline of Human Factors has developed, its scope for application to a number of new, less traditional areas has also increased. The advent of computers and microelectronics has resulted in an increase in information for an individual to take in and comprehend. Computer programs have grown so large that teams of programmers are now required, this has also brought in a number of social psychological considerations.

It is therefore important that an early integrated approach to software design and Human Factor Engineering (HFE) considerations is completed. As with any system safety analysis, it is essential that HFE considerations are addressed in the early design phases as rectifying human factor incompatibilities at a later stage is costly and often arduous.

The HFE implications of MISRA are considerable as its influence as a discipline can be seen to occur at many of the identified life cycle stages. This report will present the HFE recommendations to MISRA and also provide, in chapter one, a discussion of the guidelines development. Each section will begin with the relevant synopsis, in italic typeface, that was used in the paper presented at the 1994 Ergonomics Conference as a means of refreshing the reader's mind as to what topics were investigated and why.

Recommendations

The physical environment

- Individuals should not be expected to function effectively in poor environmental conditions. While it may be difficult to attribute poor performance solely to the physical aspects of an environment, the contributory nature of poor environmental design cannot be ruled out.

- The recommended room temperature for sedentary mental work is 21 °C. Air temperatures in winter should be between 20 and 21 °C and in summer be between 20 and 24 °C.

- Humidity of the air in a room should not fall below 30% in winter and in summer a fluctuation between 40% and 60% is acceptable.

- Draughts in a room should be kept to a minimum and should certainly not rise above 0.5 ms^{-1}.

- $30m^3$/h/person of fresh air is required for comfortable conditions and if natural ventilation is impractical, then some form of forced ventilation in the form of air conditioning should be available.

- Disruptive noise sources must be kept to a minimum, with ambient noise levels being in the range of 40 – 45 dBA.

- To combat the problems of glare, VDUs should be placed at right angles to windows with light fixtures being positioned parallel to and on either side of the operator/screen axis.

- The recommendations of the European Directive 90/270/EEC 29th May 1990 (Display Screen Equipment) which came into force on 1st January 1993, must be adhered to.

Organisation structure

- Software organisational structure should mirror design as much as is practicable.

- There should be continuous re-assessment of tasks with frequent interaction between personnel.

- Overall responsibility for a programming project should be shared and communication between project members needs to be extensive.

Error management

- Provision must be made for easy error recovery as it is virtually impossible to eliminate all errors from occurring.
- Software goals must be clear, resulting in easier understanding of potential goal conflicts and aid in the development of sub goals.
- Meaningful error messages must be employed.
- Transparent software/system design will aid error detection and planning.
- Code should be checked by individuals other than those who originally generated it.
- The maximising of recognition versus recall tasks will help to reduce cognitive errors.
- The design and use of non-similar motor sequences and the use of alternate hands for common key sequences will help to reduce motor errors.

Cognitive science

- The use of structured programming and in particular modularization will aid the development of internal semantics which in turn will aid cognitive understanding of the programming problem.
- The process of chunking can be aided by effective training of individuals. This will result in an expansion of short term memory and hence improve the cognitive abilities of programmers.
- Programs should be able to be read top to bottom to enable chunking to be completed in a sequential order.

Individual differences

- There should not be an over-reliance on the results of psychometric tests in the selection of programmers as there is no evidence that they successfully predict who will make a good programmer.
- The selection of programmers must also address behavioral factors and personality effects rather than just relying on the one dimension of intelligence.

Group effects

- Programming teams must be selected to include personalities which complement each other rather than on the sole basis of programming ability.
- Shared authority is important and it is essential that the group leader, at any given time, is the most technically competent in the particular area under investigation.
- De-centralised communication structures should be advocated where information is freely distributed between all team members.
- The product of a group's efforts should be seamless in that it must appear to be the work of one individual rather than several.
- Goal setting should involve some degree of group participation.

Job design

- Programming environments must allow for individual's voices to be heard and foster the realisation of personal and collective goals.
- Rewards systems should be tailored to individuals as the use of money as a primary motivator of individuals who function in highly technical or creative environments should not be interpreted as being the sole motivating force.
- Performance should be rewarded with skill based structures.
- Competent individuals should not be promoted out of the technical area. An alternative career structure for such individuals should be present so as to reduce the occurrence of top programming talent being promoted to more managerial type roles.

Specification format

- The specification format should perceptually mimic the solution structure.
- The format should be easily revised when changes are made to the solution.
- The syntax of a format should prohibit the construction of terms and expressions that are not allowed.

Contents

Page

1. Guidelines Development

1.1 The physical environment

The human race is amazingly adaptable and while we are able to function in a variety of less than ideal climatic situations this will be at a cost to our overall performance. It is therefore inevitable that certain aspects of the environment at work will affect an individual's ability to be productive therein. While it may be difficult to attribute poor performance solely to the physical aspects of an environment (for example poor thermal conditions, inadequate lighting etc), the contributory nature of poor environmental design cannot be ruled out.

For sedentary mental work, of which the writing of software is an example, the recommended room temperature is 21 °C.

Air temperature in winter should be between 20 and 21 °C and in summer be between 20 and 24 °C.

The mean air temperature of adjacent areas should not significantly differ from that of the air around an individual by more than 2 – 3 °C upwards or downwards and no other surface in a room, for example outside walls, should be more than 4 °C colder than the overall temperature.

Humidity of the air in a room should not fall below 30% in winter and in summer a fluctuation between 40% and 60% is acceptable

In general, air movements in excess of 0.5 ms^{-1} are considered unpleasant even if the air is warm.

Seated people will find draughts of more than 0.2 ms^{-1} unpleasant and if precise work is completed for long periods of time where an individual is motionless draughts as low as 0.1 ms^{-1} are uncomfortable. It is generally accepted that draughts at the head and knees should not exceed 0.2 ms^{-1}.

Approximately 30m^3/h/person of fresh air is required for comfortable conditions and if natural ventilation in the form of open windows is impractical because of noise pollution or poor air quality then some form of forced ventilation in the form of air conditioning should be sought.

The writing of software is an example of a "noise sensitive" activity and it is therefore important that disruptive noise sources are kept to a minimum.

Ambient noise levels should be in the range of 40 – 45 dBA.

In the case of work with VDUs it is recommended that the contrast ratio between screen and source document should not exceed a figure of 1:10 with all other surfaces in the visual environment having reflectance values lying between those of the screen and source

document.

However it is impossible to recommend one figure for the overall luminance level for an office as the level chosen is a function of differing working conditions and office size etc.

Light sources behind an individual create the risk of reflected glare while light sources in front of an individual will result in direct glare. To combat this problem VDUs should be placed at right angles to windows with light fixtures themselves being positioned parallel to and on either side of the operator/screen axis.

Light should fall at an angle of 45° to the vertical which will ensure that both screen and keyboard are in a shadowed area and hence not be susceptible to the problems of glare.

The recommendations of the European Directive 90/270/EEC 29 May 1990 (Display Screen Equipment) which came into force on 1 January 1993, must be adhered to.

1.2 Organisational structure

In terms of a working environment, the tasks of an organisation are carried out by the concerted action of many individuals and it is argued that the way in which these individuals are organised can influence how productive the organisation will be. However, despite these observations, guidelines as to the appropriate design for software organisations is under-represented in the literature. This is unfortunate, as the size of software products is increasing with a resultant effect on the size and complexity of the organisations that are responsible for them.

In organisations whose output is characterised by a high degree of creativity it is concluded that an organic type structure is desirable. This working environment is characterised by less hierarchical management structures with centre of control and communication generally being concentrated where the knowledge is. There is a continuous re-assessment of tasks with frequent interaction and overall responsibility is shared. Leadership patterns that emerge are based upon consultation and group processes with great emphasis on creativity.

Organisation structure should mirror design as much as is practicable. That is if the design indicates the necessity for a separate input subroutine then there should be a separate input group.

The most effective programming organisation structure is characterised by continuous re-assessment of tasks with frequent interaction and great emphasis on the network of authority and communication.

Management structures should be less hierarchical in nature with centres of control and communication being concentrated where the knowledge lies.

Overall responsibility should be shared and communication needs to be extensive and

widespread with diagonal and lateral as well as vertical communication patterns resulting.

The following management functions should be identified: Requirements manager, Manager of Design or Chief Architect, Software Development Manager, System Assurance and Test Group Manager, Project Manager, Administrative Manager, Technical Manager, Chief Programmer.

It is important that top programming "talent" is kept within the confines of programming and not promoted to more managerial type roles unless the organisation as a whole is able to combine both roles effectively. The promotion structure of an organisation must take into account the promotion of different types of individuals with different skills for example programmers and managers.

1.3 Human error and its management

The essence of the human factors approach to work is to design things that people use so as to enhance performance and reduce errors. It is therefore obvious that human error has important consequences for the writing of safe, reliable software. Virtually all the errors that occur in software may be considered as human errors of one type or another. Even those errors that can be attributed to hardware failures or environmental factors can be considered human errors. Errors may be introduced at every point of the development lifecycle and the main goals for any software development team are to minimise errors occurring, to reduce the negative consequences if an error does occur and to make provision for fast, effective error recovery.

In general, errors in software may be divided into four areas: design errors, coding and testing errors, operator errors and maintenance, errors with perceptual, cognitive and motor errors also occurring. While it may be virtually impossible to eliminate errors of all types from occurring it is possible to minimise the likelihood of them developing by ensuring individuals function in an environment that is conducive to the writing of software.

Aspects of this environment that are considered important are the provision of adequate work space and layout, suitable environmental conditions, relevant training and adequate supervision. Factors that are specific to individuals which must be addressed include motivation, skill level and overall job design.

Provision must be made for easy recovery from errors as it is virtually impossible to eliminate all errors from occurring.

Successful error detection is aided by clear feedback and transparent system design.

Meaningful error messages must be employed and Help Messages must directly explain how the error came about.

It is important that software goals are clear, resulting in easier understanding of potential goal

conflicts and aid in the development of sub goals etc.

In the area of information intake and integration, error management is aided when there are clear analogies and metaphors. This leads to better mental models thus producing more effective error explanations and error correction strategies.

Planning is aided by a transparent software/system design which enables an individual to develop a good mental model of the overall development and aims of the software under construction.

The most effective means of error detection is completed by separate individuals who were not involved in the writing of the original code as the most common cause of failure to detect errors is due to fixation on the part on individuals.

Cognitive errors can be reduced by maximising recognition versus recall tasks because the recognition part of the memory structure is less prone to errors than is the recall memory.

The use of consistency, rules and patterns will also aid recognition as will minimising mental calculations and transformations.

Motor errors can be reduced by designing and using non-similar motor sequences as will the use of alternate hands for common key sequences.

Error prevention by testing and monitoring is an effective means of detecting the most common errors.

1.4 Cognitive science

As a discipline cognitive psychology draws theories and techniques from several principle areas of research and it is from some of these areas that conclusions can be drawn and recommendations made for the writing of safe, reliable software.

Software development is a process based upon human skills and abilities and therefore is dependent upon how these skills are learnt, developed and controlled.

The design, development and writing of software is in essence a problem solving activity, in that a problem is posed, a strategy or plan is developed to remedy the problem and this plan is put into action. This whole process can be conveniently described as a series of human cognitive activities.

Cognitive science research into computer programming has focused upon how programmers mentally represent programming knowledge at various levels of experience. It investigates the type of knowledge and cognitive code utilised and how they are used to solve programming problems. For example a process called chunking is seen to be vital, as it is a phenomenon whereby an individual is able to combine several smaller units into one larger

unit as a means of increasing the capacity of short term memory.

As a concept it is able to be applied in a number of aspects of programmer behaviour, for example, it explains why structured programs are easier to comprehend that unstructured programs or why an individual's programming ability is language dependent; it also forms the basis of how programmers mentally conceptualise a program. While the study of individual differences may account for the largest sources of difference in programming behaviour, it is cognitive science that has provided a representation of how these differences develop and change over time.

It has been observed that it is easier for humans to learn syntactic representations for existing semantic constructs then to learn new semantic structures. Evidence for this observation is found in the fact that it is generally difficult to learn the first programming language for example, FORTRAN or PASCAL, but the learning of subsequent languages is easier. It is vital therefore that this process should be aided as much as possible in the writing of software, for example via intellectually demanding, meaningful learning.

Program comprehension is important because it is seen as a subtask of debugging and modification. It is aided by a transparent software/system design which enables an individual to develop a good mental model of the overall development and aims of the software under construction.

As a programming problem or piece of software code is being studied the reader will develop a series of internal semantics progressing from the very general to specific pieces of code focusing on minute details. There are a number of techniques that aid the development of internal semantics, for example the use of structured programming and in particular modularization.

The cognitive ability of individuals to combine several smaller units into one larger unit thereby making it easier to handle is known as chunking. Chunking is a natural progression and through experience and training individuals are able to use increasingly larger chunks of code.

Simple pieces of code are eventually embedded with others to form simple routines that are recognizable as chunks. The phenomenon provides the means to expand the capacity of the short term and working memory and as such it is a natural progression that should be encouraged.

A program should be able to be read top to bottom, as chunking is then able to be completed in a sequential order, reducing the need for reference to other parts of a program and enabling the short term memory to be concerned with one piece of code at a time.

1.5 Individual differences

The observation that the human race is diverse cannot be disputed. The study of these differences has led to a branch of psychology called differential psychology; this tries to identify and explain these perceived differences and, via a number of techniques attempts to predict the suitability of individuals for certain tasks.

Psychologists have for many years tried to identify those characteristics that they felt would be desirable or necessary in a "good" programmer. It is very apparent that programmers do differ from each other in a variety of ways and while a number of skills and attributes may be easily measured, trying to measure and quantify the mechanisms underlying the skill is problematic. It is concluded that despite at least two decades of research upon programmer selection, the study of individual differences has yet to be applied as effectively as it could be. In general, the selection of programmers has only focused upon a few mental abilities, such as intelligence, and has not fully addressed behavioral factors and personality effects.

While one of the most influential factors that has been seen to affect the writing of software is that of personality, it is almost impossible to use this variable to match individuals to certain jobs.

Personality interacts with programming success in a very subtle way. It has been often observed that in some cases, it is an individual's personality predisposition that is more important than intelligence, as intelligence is much less responsive to environment than is personality.

Personality is a dynamic rather than static attribute and selection of software engineers purely on the basis of this trait is generally unsuccessful.

There should not be an over-reliance on the results of psychometric tests in the selection of programmers as while a number of tests provide good reliable correlations when single aptitudes are tested, the testing of multiple aptitudes is still problematic.

The selection of programmers has only focused upon a few mental abilities such as intelligence and has not fully addressed behavioural factors and personality effects. This imbalance must be re-addressed.

1.6 Group effects

The traditional view of a computer programmer is one of an individual working alone. However, since most programming projects require a number of individuals, programmers are usually organised into teams and groups which results in a layer of social behaviour being imposed on the cognitive pre-requisites of the task in hand. Considerable effort therefore has been made to structure programming teams to optimise performance along basic group design principles.

The advantages of teams may be perhaps obvious in that they have an inherent ability to bring input from several individuals in solving problems. However, for their efforts to be productive it is important that the group is cohesive and their output is co-ordinated in such a way which makes it appear as the work of one individual. The software industry has struggled for many years in trying to find the most effective way to organise and manage programming teams. Two very different structures for group organisation have emerged, namely centralised and de-centralised structures.

However, it is dangerous to just assume that these two approaches adequately explain the complicated interactions that occur when a number of individuals are placed together to perform a single task. In reality few programming projects fall neatly into either of these two options. So what will result is a hybrid design where the best and most appropriate aspects from each type of structure are amalgamated into one group design, thereby making a match between the structure of both the team and the task.

Research has indicated that it is the task orientated individual who works best on projects such as software development. From this it is argued that successful programming groups are characterised by individuals who have their own idea as to how the same project should be conducted.

This has important consequences for programming group selection, as it is important to select a group where personalities complement each other rather than on the sole basis of programming ability.

In broad terms two group structures are advocated and whichever is chosen must reflect the task and the skills and personalities of the team members.

In a centralised type structure the central responsibility for programming is placed with one individual with other team members taking the roles of support staff. All technical communications should be centralised through this chief programmer and the success of this type of group design is very much dependent upon the experience and technical expertise of the individual who will take the role of chief programmer.

Centralised teams are most effective on large, simple, tightly scheduled projects that are of short duration and which do not require highly reliable or creative solutions.

De-centralised groups have shared authority with different members taking responsibility for those aspects of the work that most closely match their individual skills. This type of team is also characterised by de-centralised communication structures where information is freely distributed between all team members with the key to this egoless group’s success being the fact that the program is a shared piece of work.

De-centralised teams are best suited to tasks that are small, complex in nature that require highly reliable and creative solutions as the inherent risk associated with the task in hand is performed under relaxed time schedules over a long duration.

To be functional, research indicates that it is unwise to have a design/code split and whatever structure is chosen it is imperative that each member of the group is involved at the system design stage as individuals are then in a position to understand how their input will fit into the overall project.

The programming group must be cohesive and their output coordinated in such a way which makes it appear as the work of one individual.

The group leader of a programming team should fluctuate to accommodate the differing types of expertise that will be required at different stages of the project.

The traditional role of a leader therefore is unlikely to be successful in managing a creative group and a democratic leadership style is required as this makes the productivity of the group increase and individual members are able to work more productively.

It is important that this fact is observed by senior managers as the imposition of an unsuitable leader upon the programming team will result in tension with group loyalty being superseded by individual goals.

1.7 Job design

Working life constitutes the majority of adulthood and it is therefore not surprising that a considerable amount of research has been applied to establish what are the important factors in maximising effectiveness and satisfaction. At their most basic, theories of job design assume that both organisational and individual goals can be simultaneously met through a thorough understanding of a number of job design characteristics.

While the effects of poor job design may not be immediately apparent in the writing of software, it is another factor in the whole system equation that must be addressed. It is often the basic constituents of an individual's job that are the ones most often overlooked and it is important that all aspects that go to make up their job and place of work are appropriate.

The whole rewards system must be tailored to the individual, the organisation and its goals, with importance being placed upon motivating individuals and providing an environment that is conducive to doing so. It is often difficult to divorce the job design aspect of an organisation from the overall structure of the organisation so it is therefore important that a holistic approach is taken to designing jobs that encompass all aspects of an individual's place of work.

Design is a creative process and as such is critical to the success of large software projects. Design and managing are very different activities and it is imperative that individuals are selected for each job according to their individual aptitudes.

A number of needs must be met in an effective job design for individuals involved in writing software. These include the need: for work to be challenging and interesting, for individuals

to be involved in decision making which involves discretion and judgement, for performance to be assessed on the basis of objective outcomes, for social support in the workforce, for recognition of one's input and for a productive career path with well defined progression to career goals.

Programming environments must allow for individuals' voices to be heard and foster the realisation of personal and collective goals.

For issues of power to be adequately handled requires effective communication between managers and individuals. Therefore adequate and effective communication networks must be in place.

Rewards systems are a very important part of job design as they are a means of tying the needs of individuals to the need of the organisation. However the use of money as a primary motivator of individuals who function in highly technical or creative environments should not be interpreted as being the sole motivating force.

As the writing of software is primarily a creative, skill based exercise, it is recommended that performance is rewarded with skill based structures rather than for example, traditional piece rate monetary rewards

Competent individuals should not be promoted out of the technical arena. An alternative career structure for such individuals should be present so enabling them to remain in this area. This is especially important in organisations where top programming talent may be promoted to more managerial type roles.

1.8 Man-machine interface

Interface design is only one part of the larger process of software design and development. People have complained for many years that computer systems are difficult to use and this has led to some considerable research into the design of more user friendly interfaces. To a large extent, no matter how effective and well written the code is or how sophisticated the hardware is, if the interface is poorly designed then this will ruin what is probably an otherwise excellent system. Conversely an effective interface design can "save" poor software thus making the system acceptable to the end user.

Computing systems are becoming increasingly more interactive and an ineffective interface design will have many repercussions, for example poor system performance may result and in the extreme a system failure because of user rejection. It is therefore vital that safe, reliable software is developed using sound MMI techniques.

1.9 Risk

To explain the phenomena of risk is notoriously difficult with traditional methods being aimed

at providing numbers and calculations of riskiness in a variety of different settings, for example the nuclear and aerospace industries. Descriptive measures of risk taking tend to discuss how much and what sort of risks individuals take with little reference to why that behaviour is adopted.

For an individual to take a risk requires some thought as to the alternative choices of action as well as the potential for possible loss to the individual. A value judgement is made as to what advantages and benefits are expected by the individual together with the possible negative consequences of the decision.

Few concrete comments can be made in light of the effects of risk upon the writing of safe, reliable software. From a psychological perspective, there are a large number of influential variables that may account for an individual's accepted level of risk which will influence a number of aspects of their life, including their work. It is therefore concluded in the literature that the process of risk taking is the result of a number of underlying variables that are often unobservable and which are generally inferred by behaviour.

1.10 Specification format

There are a number of basic criteria that must be addressed by a specification, for example clarity, minimality and completeness. The specification that results from a requirements analysis is generally all the information that a design team has at their disposal. It is therefore vital that the initial specification is clear and concise as if omissions and ambiguities are present these will inevitably be carried over to the implementation.

It is important that the format chosen does not present more information than can be effectively used by the reader of the document and that the format highlights the important objectives and relationships which the author intends the reader to concentrate on. The document must present a flow of information that is similar to the flow a reader will want in processing it and the format must allow for easy abstraction to the next higher level of information. Greater emphasis is now placed upon the use of simplified and controlled use of the English language.

Diagrams produce better performance if they replace particularly complicated verbal representation as their use seems to result in the faster accessing of information.

A "nest" type organisation results in better performance that a list with "go to" type statements.

Flowcharts are useful in aiding decision making tasks but they are not as useful as other formats (such as constrained languages) for programming tasks.

Flowcharts offer some help to novice programmers as they help individuals to follow the decision structure of the program.

An effective specification format is one which highlights relevant information that is useful to the reader.

The syntax of a format should prohibit the construction of terms and expressions that are not allowed.

Perceptual and symbolic aspects of a format should present and highlight the same information.

The format should perceptually mimic the solution structure.

The format should be easily revised when changes are made to the solution.

2. Reference List

Differential Psychology, A.Anastasi, The MacMillan Company, NY, 1965.

Cognitive Psychology, J.B.Best, West Publishing Company, LA, 1986.

The Development of Job Design Theories and Techniques, W.A.Buchanan, Saxon House, London, 1979.

Health and Safety (Display Screen Equipment) Regulations 1992: Display Screen Equipment Work: Guidance on Regulations, Health and Safety Executive.

Human Error, J.Reason, Cambridge University Press, 1990.

Handbook of Human Factors, G.Salvendy, Wiley and Sons, NY, 1987.

The Physical Environment at Work, D.J.Oborne M.M.Gruneberg, Wiley and Sons, London, 1983.

The Small Group, M.S.Olmstead, Random House NY, 1959.

The Psychology of Computer Programming, G.M.Weinberg, Van Nostrand, Hold Rein NY, 1971.

3. Glossary

Cognitive science: The study of how sensory input is transformed,reduced, elaborated, stored, recovered and used.

Glare: The overloading of the adaptation process of the eye as a result

of over exposure of the retina to light.

Human error: Inappropriate human behaviour system efficiency, safety and performance.

HFE: The application of scientific knowledge about human begins to produce a product, job or environment which considers the abilities, limitations and responses of the people who will use it.

Individual differences: The study of variability in a number of human traits and abilities.

Job design: Those aspects that go to make up an individuals' terms of employment.

Man machine interface: The boundary between human, the environment and system.

Motivation: The forces or influences that provide incentives for behaviour.

Organisational structure: The physical and philosophical working environment an individual finds themselves to be part of.

Physical environment: Climatic aspects of an environment such as heating, lighting and noise.